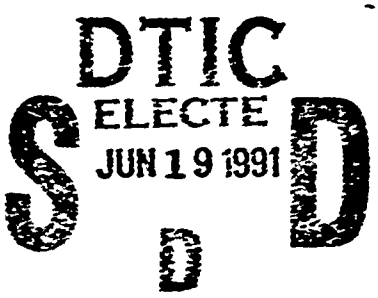

NW Lab for Integrated Systems

FINAL REPORT

24 May 1991

NAME OF CONTRACTOR:	University of Washington Department of Computer Science, FR-35 Seattle, Washington 98195
EFFECTIVE DATE OF CONTRACT:	23 February 1988
DATE OF PRE-CONTRACT AWARD:	5 February 1988
CONTRACT EXPIRATION DATE:	30 September 1990
CONTRACT NUMBER:	N00014-88-K-0453
PRINCIPAL INVESTIGATOR:	Larry Snyder (206) 543-9265
TITLE OF PROJECT:	VLSI Architectures & CAD
CONTRACT PERIOD COVERED BY REPORT:	5 February 1988 - 30 September 1990

Sponsored by
Defense Advanced Research Projects Agency (DoD)
Issued by Office of Naval Research
Under Contract #N00014-88-K-0453

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency of the U.S. Government.

Introduction

This report gives an overview of the research and development projects supported by the "VLSI Architectures and CAD" contract between DARPA and the University of Washington's Northwest Laboratory for Integrated Systems. Because most of these scientific and engineering accomplishments have been reported on in the technical literature, this report serves primarily as a introduction to the topics. Emphasis here is on overall directions, motivation, and describing what has been achieved. References are provided to the appropriate papers, where the full details can be found.

As the name "VLSI Architectures and CAD" suggests, the contract addressed many topics that generally fall into the architecture related topics and computer aided design topics. A table of contents is shown on the next page. At the end of this report is a list of the principal technical papers produced under the contract.

Table of Contents

The APEX Prototype

Robert Bedichek, Tony DeRose, Carl Ebeling, George Winkenbach

In the near future commercially available graphics workstations will be capable of real-time display of high quality shaded images of relatively complex environments. These systems typically devote a good deal of special purpose hardware to the transformation, clipping and shading of polygonal models. Such systems are proving valuable for applications such as engineering design and scientific visualization. However, these systems are still very expensive, and they are not well suited for non-polygonal models such as those arising in spline-based design.

We have recently constructed a much less costly implementation for supporting spline-based engineering design applications. Rather than opting for a large amount of custom hardware to accelerate the entire image generation pipeline, we have instead chosen to focus on the construction of a small amount of carefully chosen hardware to support lower quality display. The use of low quality imagery is not, in many cases, a serious hindrance. In many geometric design applications it is generally satisfactory to use low quality real-time images to convey the basic shape of an object while the designer is performing a deformation such as the movement of a control point. Molecular modeling systems, for instance, have used "dot representations" of complex surfaces for some time. A dot representation is simply a relatively sparse collection of points lying on the surface of the object. When used in conjunction with depth cuing and stereoscopic viewing, dot representations are an effective means of communicating shape without the additional complications of hidden surface removal and complex shading calculations. During more static phases of the design process when subtle shape variations must be assessed, slower, higher quality images can be computed using little or no hardware acceleration.

The Apex geometric design workstation is built around a single VLSI chip which generates a wide range of parameter curves including Bézier and uniform and non-uniform B-splines based on an elegant method due to de Casteljau. This algorithm for a cubic Bézier curve results in a three-level labeled digraph with a regular triangular interconnection. This computation graph, called a *triangular computation* forms a data-flow graph which can be mapped directly into hardware. In general, computation of a point on a Bézier curve of degree d can be viewed as a triangular data flow graph with d levels. This algorithm can also be generalized to encompass other curve schemes, including B-splines, cubic Catmull-Rom curves, Pólya curves, and Lagrange curves.

We investigated two architectural methods for implementing the triangular computation. [DeBaBa*89] The first used a single high-performance processor optimized for the triangular computation. The second was a special-purpose architecture which implemented the triangular computation directly in silicon. We showed that by taking advantage of the structure

of the computation, we could achieve much higher performance in a special-purpose architecture than in the more general-purpose processor implementation. This special-purpose architecture is the basis for the Apex I chip which implements a degree three triangular computation and supports surface generation by computing three co-ordinate values simultaneously.

The completed chip contains 60,000 transistors and was fabricated by MOSIS in 2 micron CMOS using a 7mm x 9mm 84-pin standard frame. We experienced a yield of 60% in functional testing. The chip uses a 10MHz clock and performs 30 million multiplications and 180 million add/subtracts per second (16-bit numbers). At this clock rate, the chip can draw 3-D points at a sustained rate of 3.33 million points per second.

We have now completed a prototype geometric design workstation comprising a Macintosh II host processor, an Apex processor board, and a Tektronics stereoscopic frame buffer and monitor. [BeEbWiDe91] These work together to convert a control net description of a curve or surface into a 3-D view on the stereo monitor. The host software is responsible for initializing the Apex registers for each curve or surface by downloading control points and label registers. Once initialized, the Apex board computes the x, y, and z co-ordinates of points lying on the curve or surface using the Apex chip. A stereo pipeline on the Apex board then performs the stereoscopic perspective calculation to generate the x,y co-ordinates for the left and right eye views, and writes this data into the stereo frame buffer over the NuBus.

We have measured the performance of this workstation using a realistic imaging task and obtained a rate of 200,000 points per second. While this is acceptable for interactive design of reasonably complex objects, it is well below that achievable by the Apex chip. The performance of this prototype workstation is limited primarily by the speed of the NuBus and the frame buffer. Integrating the Apex chip directly into the frame buffer would remove this communicatin bottleneck and allow points to be drawn at the maximum frame buffer rate.

The decision to map the triangular computation directly into hardware allows a simple, high performance implementation, but it is neither flexible nor scalable. That is, it restricts the degrec of the curve generated to the size of the implemented triangular computation, and if larger triangles are desired, either for performance or flexibility, it does not extend readily to a multi-chip implementation because of the problem of partitioning two-dimensional arrays like the Apex across several chips. We have described an alternative mapping [BeEbWiDe91] that uses a set of uncommitted physical processors and dynamically schedules the triangular computation onto these processors according to the degree of the curve being generated. Since the triangular computation is fixed, the best schedule for each degree curve can be pre-computed and the processors and communication paths specialized to that required by the set of schedules. Thus with a slight increase in the flexibility of each processor and the communication paths the processors on a single chip can be made to generate curves of

varying degree with little or no loss in performance.

We have defined a second-generation Apex chip that uses this flexible computation structure along with several other modifications that have resulted from our experience with the prototype Apex workstation. These include extended support for surface generation, perspective transformation on-chip, and increased data precision. We project that such a chip, comprising about 150,000 transistors would operate at 50 MHz and generate 3-D surfaces directly at the rate of 16 million points per second, corresponding to over 1 billion operations/second. This single Apex chip would saturate all but the most expensive frame buffers.

Hypercube Routing

Smaragda Konstantinidou and Larry Snyder

Two substantial router studies were conducted with support of the contract: Invention of the Chaos Router and Invention of the Zenith Router. Before describing these two projects, some basic concepts about hypercube routing must be covered.

A hypercube of dimension d has 2^d nodes each of degree d. Nodes are assigned an integer address in the range $[0,2^d-1]$ and node n is connected to node m if the binary representations of n and m differ by one bit position. This means that a message can start at source node S and be routed to destination node D by simply following a path that "changes" the bits of S into the bits of D, i.e. changes the positions that are "1" in S XOR D. The total number of edges that must be traversed is given by the Hamming distance of S and D.

Continuous routing, the case considered here, refers to networks that accept and deliver messages continuously; the alternative is a batched routing policy. *Oblivious* routers send messages along a single path between source S and destination D. The alternative is *adaptive* routers that (incrementally) select a congestion-free path for messages from among the available alternatives. An adaptive router is said to be *minimal* if it only sends messages along shortest paths between the source and destination. Intuitively, minimal routers only go forward. Alternatively, *nonminimal* routers send messages along arbitrary paths.

A *deadlock* is said to exist in a routing network if one or more messages are stopped and cannot move again for any future (normal) behavior of the network. A *livelock* is said to exist in a (nonminimal) routing network if one or more messages continually circulate without being delivered. *Starvation* is said to exist in a network if one or more nodes are continuously prevented from injecting a message. When these conditions do not occur, the router is said to be deadlock-free, livelock-free and starvation-free, respectively.

Chaos Router

The Chaos Router is a randomizing, nonminimal adaptive router developed by Konstantinidou and Snyder [KoSn90]. It has been shown to be deadlock-free, probabilistically livelock-free and probabilistically starvation-free. Further, it has initiated considerable research that is not covered by the present contract.

Adaptive routers like the Zenith Router and the Chaos Router have often been advocated as replacements for the state-of-the-art oblivious routers used in commercial machines because adaptive routers can by-pass congestion, speeding delivery, and they are fault tolerant. There have been many adaptive designs, but none have prevailed. Though they are in principle the fastest, nonminimal routers require livelock protection to assure that the messages are

always delivered. This slows their circuitry. Since routers are always compared in a fully operational network with no traffic where the advantages of an adaptive router are not visible, the adaptive routers always lose to the oblivious routers. The Chaos Router, however, is the first nonminimal adaptive router to solve this complexity problem.

The Chaos Router applies the following set of rules:

- Messages are sent along a minimum, congestion-free path to their destinations as long as such a path can be (incrementally) found.
- If a message arrives at a node for which all forward links are blocked, the message waits in a local buffer until a forward link becomes free, provided there is buffer space.
- If there is no buffer space, a waiting message is randomly selected and sent out of the next available channel.

This latter activity is called *derouting* because the message is likely to be sent away from its destination.

The Chaos Router solves the "complexity problem" for nonminimal adaptive routers by eliminating livelock protection. It can avoid the need for livelock protection because of the random choice of which message to deroute. Thus, unlike other adaptive routers where the deterministic choice allows messages to get "locked" into persistent cycles preventing them from being delivered, the Chaos Router's random choice makes it unlikely for the cycles to persist for long. The Chaos Router has been shown to be *probabilistically livelock-free*, i.e. the probability that a message remains in the network t seconds goes to zero as t increases. In practice, since probabilistic livelock freedom is operationally equivalent to deterministic livelock freedom, the Chaos Router does not need any protection.

In addition to showing probabilistic livelock freedom, the Chaos Router has been shown to be deadlock-free and probabilistically starvation-free. Preliminary simulation results were also compiled indicating that it is not only faster than a lifelock protected router, but also does less derouting.

Zenith Router

The Zenith Router is a deterministic, minimal, adaptive router developed by Smaragda Konstantinidou [Ko91, Ko90]. She showed that it is deterministically deadlock and starvation-free. Since it is minimal, it is automatically livelock-free. Additionally, she performed simulations to determine the optimal queue size (8 messages), and average and worstcase latencies for various sizes of hypercubes.

Towards describing the zenith router, consider the following hypothetical router. A message sent from source S to destination D is said to have as its zenith the node (S OR D). If the bits of S that must be changed to reach D are classified as 0-to-1 changes and 1-to-0 changes, then any message can be routed through its zenith by first making the 0-to-1 changes in any order followed by making the 1-to-0 changes. If all forward links are blocked by traffic a message waits in a queue until a forward link is free. Deadlock is avoided in such a router by having two queues, one for "upward" moving messages and one for "downward" moving messages. (Messages are in the *ascend* phase or the *descend* phase when they are making 0-to-1 or 1-to-0 changes.) The problem with such a scheme is that the node labeled 2^d-1 and the nodes around it are bottlenecks since many messages must pass through these few nodes.

This router can be improved. Define the *nadir* between nodes S and D as (S AND D). The messages could be classed into two groups, class 1 messages begin with the ascend phase and end with the descend phase, passing through the zenith, and class 2 messages begin with the descend phase and end with the ascend phase, passing through the nadir. This reduces the traffic at the "top" of the network at the expense of introducing a new bottleneck at the "bottom" of the network. Interestingly, deadlock can be prevented using only three queues per node. Class 1 messages use queue 0 for the ascend phase and queue 1 for the descend phase. Class 2 messages use queue 1 for the descend phase and queue 2 for the ascend phase.

The Zenith Router employs these ideas adaptively. All messages are injected as class 1 messages, and thus enter queue 0. They are routed as class 1 messages as long as they can continue to make forward progress. When all forward paths are blocked, say at node P, and queue 1 is not full, then the message switches to being a class 2 message by being moved to queue 1. It will pass through the nadir of nodes P and D, and continue on to its destination with an ascend phase.

Konstantinidou [Ko91] proved that the Zenith router is deadlock-free and starvation-free, she gave a detailed design including channel protocols and she simulated its performance for various network sizes. The zenith router is believed to be the fastest known minimal router.

Asynchronous Channel Design

Bill Barnard and Kevin Bolding

A communications channel suitable for use in a Chaos Router has been designed by Bill Barnard and Kevin Bolding. The circuits at each end of the channel have an input frame and an output frame each capable of holding a 128-bit message. The circuits exchange "ownership" of the connecting "wire" long enough to transmit a single message in 8-bit flits. The asynchronous design was fabricated during spring of 1990.

VLSI-Oriented Data Compression

Suzanne Bunton and Gaetano Borriello

We empirically evaluated the combined effects of dictionary size and a variety of replacement strategies on the effectiveness of Ziv-Lempel encoders. Our results demonstrated that LRU (least recently used word) replacement strategies most significantly reduce dictionary size requirements in practical Ziv-Lempel encoders, without compromising compression. However, the additional overhead of traditional LRU implementations makes them impractical for use in fast hardware compressors. As a solution, we introduced a new dictionary management heuristic, "TAG", which provides the compression advantage of LRU schemes, while requiring only a fraction of the additional memory and computational resources of LRU implementations. A practical CAM-based 2μm CMOS implementation exploits the simplicity and regularity of our design to achieve a fixed 20MB/sec data rate. Thus, our Ziv-Lempel implementation realizes a speed improvement of 10 to 20 times that of the fastest recent hardware data compressors. [BuBo89, BuBo90, BuBo91]

The data compression scheme of Ziv and Lempel repeatedly matches the input stream to words contained in a dictionary and returns pointers to the locations in the dictionary of the longest matches. Initially, the dictionary contains only the single character strings over the input alphabet. These initial dictionary elements are permanent to ensure lossless compression. After every match, the matched word concatenated with the next symbol of the remaining input stream and is added to the dictionary. This process continues until the input stream is exhausted. The dictionary growth heuristic implied by the addition of the last parsed word concatenated with the first unmatched symbol causes the dictionary to contain every prefix of every word it holds. Thus, a trie is the natural data structure for real-time parsing. Typical implementations represent the trie as a table where each entry consists of a pointer to a word's longest proper prefix (parent) and the word's last character. Realistic implementations have finite memory, so dictionary growth is bounded. Since dictionaries that adapt continuously generally provide better compression, we needed an efficient dictionary management scheme. "TAG" was developed to address this need.

The primary consideration in its design was that the replacement policy implementation must be parallel to the basic compression process. This is achieved by tagging trie nodes with a value that can be used to determine which leaf node has been visited least recently and can be reused for a more current word.

Our Ziv-Lempel variant is demonstrably superior in compression performance (achieving at least as good compression as any other Ziv-Lempel and requiring less memory due its superior adaptivity. The algorithm translates naturally to a hardware implementation, which has been designed in 2μm CMOS at the transistor level, simulated using SPLICE, and is ready for fabrication. Our throughput estimates are 10 to 20 times those of HP's Ziv-

Lempel-hardware and IBM's Q-Coder chip, yet our proposed implementations are no more ambitious than these designs, which have been successfully prototyped. Furthermore, as our design uses a fixed-width output code and can input a new character every clock cycle it minimizes system integration problems.

MacTester - A Low-Cost Functional Chip Tester

Carl Ebeling and Neil McKenzie

Testing VLSI chips in the University research environment is typically done in an ad hoc way. Some sort of hardware environment is designed around the chip to create a self-contained system which can be operated, observed and debugged using a logic analyzer or oscilloscope. Testing and debugging a partially completed system or a subsystem that operates in a complex environment can be very difficult. As the complexity of chips grows, the problem of testing is becoming even more difficult. Although commercial testers provide the necessary functionality, their cost is prohibitive and the usual testing environment is primitive. Moreover, in most cases their functionality far exceeds that required for system prototyping.

The goal of the MacTester [EbMc90] is to remedy this situation by providing a powerful, interactive environment for functionally testing VLSI chips. Functional testing means that the *values* and *sequencing* of the I/O signals can be verified but that precise *timing* cannot. While speed testing is clearly important, functional testing and debugging concerns tend to dominate in custom chip design and the issues of functional testing and debugging can be separated from those of timing validation. Moreover, the MacTester supports a limited version of speed testing and full speed testing can be done by adding a small amount of prototyping hardware to the MacTester setup.

The overall goal of the MacTester was high functionality at low cost. The MacTester has the following characteristics:

- A total of 128 test signals are provided so that large chips (up to 132 pin PGAs) can be accomodated.
- Test signals are bi-directional and can be individually tri-stated on a vector by vector basis. Test signal direction is specified completely by the test program or set of test vectors.
- The test head is a 20x20 ZIF socket that accomodates DIP packages up to 40 pins (64 pin DIPs require a special header) and PGA packages up to 132 pins. This test head makes test setup very simple and allows the tester to be shared by several projects.
- On-board headers make every test signal accessible for observation using an oscilloscope or logic analyzer, or for connecting the chip to external circuitry. These headers can also be used to supply power and ground to the chip, although the on-board signal buffers can provide sufficient current to power typical CMOS chips for functional test.

- Provision has been made for dynamic circuits by including an on-board memory capable of storing several thousand test vectors. The tester can independently present these vectors at a rate of 1 MHz.

The tester software provides a simple, intuitive interface that makes it easy for programs to use the tester. These programs range from simple test programs devised by the user to sophisticated interactive programs that supply a visual interface via schematics and timing diagrams.

The MacTester implementation makes extensive use of Xilinx FPGAs for the data path and control circuitry of the tester. These FPGAs proved to be both cost-effective and a flexible implementation medium. The MacTester fits on a single 8" x 14" PC board with a total parts cost of under $1000.

Five MacTesters have been contructed thus far and we are currently trying to transfer this technology to a company that will make it available to University research and instructional laboratories at low cost.

Models of Computer Architecture

Sam Ho

Though there have been theoretical models of computers predating the creation of the electronic computer, these have not been helpful to computer architects because they are too abstract. At the other end of the spectrum are analytical models of computers used by machine designers to understand the performance characteristics of a particular computer. These are not useful to architects either because they are too specialized. There have been few intermediate models that describe the general principles on which all architectures must be built. Sam Ho studied this problem and created such a model called the *Normalized Time Model.*

Normalized time is a cost/benefit model that generalized earlier work at UW by Holman. The essential idea is to begin with a base architecture, describing a particular set of capabilities, and consider alternatives to improvement only to the extent that they "pay their way," i.e. that the benefits (increased speed) returned for the costs (increased area, transistors, power, etc.) improve the base architecture more than any other alternative.

Using the normalized time model, for example, Ho validated the intuition that for almost all computations the addition of a floating point unit to a (parallel) processor is worth the cost, i.e. the factor of improvement per transistor added is greater than 1. Alternatively, a full 32-bit shift register is not, since not enough instructions in the typical stream benefit from the large increase in area.

Recognizing that architectural components such as multipliers and shifters need not be added on an all or nothing basis since one can have small versions (e.g. 16-bit) that are repeatedly used, Ho parameterized the model to find the optimal size. He found, for example, that the optimum size for a shifter on Berkeley's RISC II is 10 bits. These parameterizations could also be used to analyze the size of the data path, and the study lead to the conclusion that the ideal size for a VLIW machine in current technology is perhaps 3 words wide.

Ho used these concepts to define a new computer architecture he called the *asymmetry machine.* The principal idea is to add processors to each of the memory boards to permit computations with pages concentrated on that board to execute concurrently with the primary processor. These processors added in a way that limited their architectural impact, and therefore the costs, are intended to provide limited parallel capability. The analysis indicated that perhaps five such processor enhanced memory boards would be beneficial with today's technology.

Additionally, Ho developed a model of the processor/memory interface which is applicable not only to sequential processors, but to SIMD parallel processors as well. The motivating question was to clarify the argument between architects as to which is the proper approach,

word-parallel computers or bit-serial computers. The results, which depend on a simple model of memory reference, indicate that over a broad range of algorithmic assumptions, it doesn't matter much. If there is sufficient parallelism in the computation to hide the latency, the essential problem is the bandwidth limitations. If not, then the problem is very complex; but it is not primarily a memory organization problem. Ho used the model to analyze empirical results on bus organization, finding his model to be effective at predicting the system behavior.

Hardware Assist for Performance Evaluation of Multi-processors

Craig Anderson, Kathy Armstrong, and Gaetano Borriello

The purpose of this project was to develop a highly configurable VLSI device that could be used for instrumentation applications and, specifically, the collection of event traces on multi-processors. The chip consisted of a large content-addressable memory that could be programmed to check for particular conditions on the wires being monitored. The results of the match could be used to: increment counters (to count events), generate information to be collected (to trace events), and/or set internal state bits (so that complex compound events could be detected). [AnArBo89]

The primary considerations in our design were that it be expandable when more wires required monitoring or more events needed to be detected and counted or traced. Also, the design included features that would allow multiple devices to collect data from different sources but merge their collected traces into a single stream for data compression and storage for later analysis. This last feature is crucial for the monitoring of multi-processors.

In our final design, each chip included 64 separate 60-bit wide event comparators. Of the 60 bits, 32 are used to monitor system signals while the other 28 are used to check detecting complex events by looking at the values of internal counters and state. In this manner, compound events can also be detected. For example, that a signal went high, followed some arbitrary time later by another, and then another that must have changed some specified number of cycles later. The chip also included 24 counters for collecting histograms of events and logic to generate time-stamped traces.

Reconfigurable Logic Arrays

Carl Ebeling, Robert Condon and Bill Barnard

We have defined the Reconfigurable Logic Array (RLA) as a solution to the problem of computing complex evaluation functions quickly and efficiently using reprogrammable, reconfigurable special-purpose hardware. [Eb91] The RLA contains two major components: a state vector and an evaluation tree. The state vector captures the information about the current problem state that is required for the evaluation. The evaluation tree maps an instance of the problem state contained in the state vector to the desired value.

We first explored a large design space including both custom VLSI circuits and standard off-the-shelf components to determine the relative advantages of different alternatives. As part of this exploration, we examined in detail the computational requirements of a wide range of chess evaluation functions and their implications. We used the results of this study to design, fabricate and test a set of general, parameterized components from which evaluation hardware of various types can be constructed. Finally, we attempted to determine the generality of this evaluation architecture. This was done first by analyzing the restrictions inherent in the state vector and the evaluation tree, showing the extent to which these can and cannot be removed.

The state vector component of the RLA maintains the information relevant to the evaluation in a set of programmable registers. This information can be maintained a number of different ways depending on the characteristics of the evaluation. We designed a set of state registers that cover a broad range of problem characteristics and implementation alternatives. These have been analyzed in terms of cost and effectiveness for a range of potential update modes and evaluation characteristics. One important result is demonstrating how evaluation computation can be moved from the evaluation tree into the state vector under certain conditions. We also analyzed these conditions and showed different ways to perform updates to make this computation possible. This work has led to a much more efficient design of the state vector that can be implemented either as part of a custom evaluation chip or using off-the-shelf parts. It has also led to a family of state register designs that can be "plugged into" a variety of problems.

The evaluation part of the RLA relies on a tree of general function blocks, each of which computes a mapping function over its inputs. The information held in the state vector becomes the input to the function blocks at the base of the tree. These function blocks analyze this information, mapping it into categories of smaller size. The function blocks at subsequent levels of the tree map this data further until a final value is achieved.

We analyzed the relative effectiveness of RAM and PLA implementations for the function blocks to determine the most efficient implementation of the evaluation tree. Since dynamically reprogrammable PLAs are not commercially available, we designed an efficient

reprogrammable PLA structure which we call an rPLA. We compared the use of RAM and PLAs for the function blocks by mapping a set of general chess evaluations to a range of different evaluation tree structures. The surprising result was that a combination of these two methods provided the most efficient implementation. The rPLA's are used near the leaves of the tree, while RAM is used near the root. We also showed that when more restricted state updates must be used, the rPLA implementation becomes very effective since evaluation that could otherwise be moved to the state vector now must be done in the tree itself. This study also explained why a complete RAM implementation of the evaluation tree is an acceptable alternative for chess based on the fact that the incremental updates used in chess allow very general computation in the state vector. We conjecture that this will not be true for other problems.

We designed, fabricated and tested a set of VLSI modules for use in constructing a state vector and evaluation tree. These modules are programmable so that different evaluations can be computed by defining different functions for the function blocks. They are reconfigurable in the sense that the state registers can be reconfigured to capture different information. They are parameterizable in that the size and characteristics of the modules can be redefined when a chip is constructed.

We successfully fabricated and tested these modules to validate their functionality and performance. Our test chips showed good yield and the performance results were consistent with our simulation results.

Fully-testable CMOS Asynchronous Counter

Jerry Carson and Gaetano Borriello

We explored testing issues in asynchronous design by implementing a fully-testable asynchronous counter. Our approach was to perform a study similar to one that had been had done for a synchronous counter and evaluate the cost of extra testing logic in terms of performance degradation and area penalty. Finally, our study also considered the time required to test the counter. [CaBo89, CaBo90]

We designed an asynchronous n-bit CMOS counter with a scan path to access internal state has had been done for the synchronous counter. The counter is thereby converted into an n-bit "master-slave" asynchronous shift register with the counter's *req* input also being used as the shift register input. The only observable outputs are the *ack* and *carry-out* signals. The counter utilizes two-cycle (transition) signaling and guarantees that new output values are available before *ack* is toggled. Two 16 bit counters, one base design and one scan-based design have been fabricated on the same chip (2.0 μm n-well CMOS) thru MOSIS. There are three control signals required for the testable counter as opposed to one reset signal for the base counter. However, the three signals may be wired together after testing is complete and used as a reset signal for the testable counter.

Four parts were received; all of which passed the test suites developed. Speed testing of the parts shows a count rate of 22.2MHz for the base design counter, versus 21.0 MHz for the scan design, indicating a speed degradation of 5.8% for the testable design as compared to the base design. The testable design is achieved with a 15% increase in transistor count (from 52 to 60 per bit), and a corresponding increase in chip area of approximately 6%.

Our test vectors checked for all stuck-at and stuck-open faults in the counter designs. Interestingly, we showed that test time is $O(n)$, when the counter outputs are not observable, compared to $O(n^2)$ time for a synchronous counter. The asynchronous counter has twice as many state bits as the synchronous counter due to the extra state required to handle the more complex asynchronous state changes. The result of $O(n)$ showed that this extra state can be taken exploited to dramatically reduce testing time. The two state bits in each cell may be set independently, and it is the extra state bit that allows testing in $O(n)$ time since carry propagation chains may be started/restarted anywhere in the counter by appropriately setting the pairs of bits in each cell. It remains to be shown whether the extra complexity of asynchronous logic can be similarly exploited in reducing testing time in the general case.

Addressable Memory for "Fuzzy" Matches

Bill Barnard

A content addressable memory (CAM), a circuit which when given a target vector finds the closest match (i.e. smallest Hamming distance) among a set of previously stored vectors, has been designed with the ability to match "fuzzy" or partially known data. The purpose of such a project was twofold. First, it was thought that the fuzzy CAM would provide an especially powerful building block for implementation of neural networks. Second, the mixed analog/digital nature of the fuzzy CAM allowed us to test our Network C design tool (see Section on Network C).

An original design with 16 vectors of 48 bits each was created using our standard design tools. The design was expressed in CFL; the digital portion was simulated using our standard simulator RNL; and the analog portions were simulated with SPICE, since the full circuit was too large to permit complete simulation. This chip was fabricated and tested during the summer of 1988, but it was only partially functional due to a bug in the dynamic refresh logic.

In the redesign, extensive use was made of WIN — the Washington Intermediate Notation (see Section on WIN) — as well as Network C. In addition to providing a tighter coupling between the layout and the network description, this recasting had the advantage of simplifying the generator code; source code for the WIN generator was a factor of 4 shorter than the original CFL generator. Effective though it was as a test vehicle for our CAD tools, this second design had a bug in it as well.

WireC: Combining Graphics and Procedures to Describe Circuits

Larry McMurchie, Zhanbing Wu and Carl Ebeling

WireC is a specification language that provides both the clarity of graphical representations and the expressiveness of procedural constructs for the description of complex electronic systems. Graphics allows fast, intuitive interaction with a design description while procedural constructs give the power and flexibility to describe circuit structures algorithmically. Procedures also allow parameterized designs so that a single description can represent a whole family of devices. These two types of description are complementary and WireC gives the designer the freedom to choose the most appropriate representation. Since C constructs can be embedded in schematics and schematics embedded in a C program, this choice can be made at a very fine grain.

Behavioral descriptions are easily incorporated in WireC allowing the designer to mix structural and behavioral representations. This is done by allowing WireC devices to reference descriptions in other languages. Typical design descriptions use WireC to describe the structure of the system and use behavioral and logic descriptions written in languages like MEG and BDSYN to describe finite state machines and random logic. WireC then drives the synthesis programs to generate a composite circuit. Thus WireC is a framework that can take advantage of a variety of description languages and synthesis tools.

WireC schematic programs can be compiled into different output forms by choosing the appropriate library. A library contains all the available primitive devices for a particular technology. The library also specifies the output format to be produced when a specification is compiled. Although the CMOS library is the one that is most widely used, we have also compiled libraries for off-the-shelf TTL components as well as a library for designing Xilinx FPGAs.

WireC, and its predecessor WireLisp, have been used to design a number of substantial VLSI chips including the Apex graphics chip and an RLA test chip. [EbWu89] These are large designs containing 60,000 and 120,000 transistors respectively. WireC compiles over 1000 devices per second on a Sparcstation 2 so that even a large design with 100,000 devices takes less than two minutes to compile. WireC has been distributed to a number of other Universities including the University of Wisconsin, Stanford, Santa Cruz and Penn State.

Timing Optimization of Multi-Phase Logic

Karen Bartlett, Gaetano Borriello, and Sitaram Raju

High-performance MOS circuits are frequently designed using precharged and dynamic logic. This requires the use of multiple phases of the system clock to ensure that the circuitry is precharged and refreshed at the proper times during each clock cycle. Finite-state machines used to control this type of logic must therefore be constructed as multi-phase sequential logic with inputs and outputs stable during the appropriate phases. The timing optimization of multi-phase logic entails the reduction of the overall cycle time of the machine and/or input to output delays by distributing computation throughout the entire clock cycle. We have developed a tool to automatically perform this optimization task and have implemented it as a set of extensions to the combinational logic optimization tool, misII. Our algorithms yield improvements that are on average 10-20% better than what is achievable using purely combinational logic optimization tools that do not move logic across latches. These improvements represent 75% of what would be possible in the most idealized case. Results on simple two-phase circuits show average input to output delay improvements of 13% with area penalties of 11%. For a four-phase controller used in the SPUR processor it yields an improvement in cycle time of 18% with an area penalty of 11%. Furthermore, our experiments indicate that the optimization algorithm is highly insensitive to parameter variations in the underlying combinational logic optimization routines and initial state assignment. [BaBoRa89, BaBoR^0, BaBoRa91, BuSaBaBo91]

The basic idea behind our approach is that of guiding the movement of logic across latch boundaries so as to minimize the length of the critical paths in the circuit. This is accomplished by decomposing the circuit into blocks between adjacent clock phases. A goal delay is then determined for each block that corresponds to the best possible case where as much logic as possible is moved off the critical path between primary inputs and primary outputs of the block. This is an overly optimistic goal in that it assumes that inter-block (state) signals will never be on the critical path. However, it does give the algorithm an indication of where there is the most to be gained. Logic movement and resynthesis operations are then used to move logic off the head and tail of the critical path of a block and into another block (hopefully, onto a non-critical path). The algorithm makes tradeoffs to mitigate the greediness of the approach and makes some detrimental moves in the hope that there will later be larger improvements.

Hybrid Compiled/Interpreted Switch-level Simulation

Craig Anderson, Gaetano Borriello, and Larry McMurchie

We have developed a switch-level simulator that combines the speed of compiled simulation algorithms with the flexibility and fast set-up times of interpretive schemes. Compiled simulation is fast for simple subcircuits and slow for certain complex ones. Interpretive schemes have a fast set-up time, but are slow in simulating simple circuits because of the overhead of the interpreted data structures. Our hybrid scheme, based on COSMOS and MOSSIM II, offers the best of both approaches, reducing simulation time for a variety of common circuits and saving a factor of 3 or more in set-up time. [McAnBo91]

Simulators that perform a switch-level analysis of MOS transistor networks have existed since the late 1970's and are widely used for the verification of MOS integrated circuits. Switch-level simulators must correctly model a wide range of effects common in MOS circuits (e.g., dynamic storage and charge sharing) and all the design methodologies available to MOS designers (e.g., complementary, ratioed, pre-charged, domino, and pass transistor logic).

MOSSIM II was one of the earliest switch-level simulators to address all these issues. The MOSSIM algorithm performs an interpreted analysis of the transistor network to determine how changes in node values propagate. This is a very flexible approach permitting the modification of the circuit being simulated at run time. However, most of the circuit structure usually is not changed and this is not exploited in optimizing simulation speed. Recent efforts in interpreted simulation have made the analysis of the circuit more efficient by refining the data structures and graph traversal schemes.

The compiled simulation approach was developed as an attempt to improve the performance of switch-level simulation. In the COSMOS system, a set of Boolean formulas that describe the mapping of input and current circuit state to output and new circuit state are extracted from the transistor network. By compiling the equations into an executable file, simulation time can be dramatically improved. However, the modify-simulate-debug loop is slowed by the computation required in partitioning the initial transistor network into channel graphs and the Boolean analysis of those channel graphs.

Interpretive analysis schemes such as MOSSIM suffer from the high overhead of traversing small channel graphs, such as simple gates. For these types of graphs, the minimal computation required by the compiled approach is clearly advantageous. On the other hand, interpretive schemes have a clear advantage for large channel graphs, only a small part of which are active at one time. By virtue of the interpretive analysis of the channel graph, such schemes can isolate the active parts of a large graph and solve for the steady state. Therefore simulation time is mostly dependent on the subset of nodes that are changing. Compiled approaches must solve for the values of all the nodes in the channel graph whether

or not they will actually change state. Bryant has analyzed the size of the equations resulting from an n-node network and has determined that in the worst case the resulting Boolean formulas may contain $O(n^3)$ operations, although most commonly found structures contain only $O(n)$ operations. Compiled approaches suffer from another disadvantage. Consider a large network composed of numerous channel graphs. Even a small localized design change necessitates a reanalysis of the entire circuit. Because the reanalysis time is substantial, these simulators have been modified to recognize previously analyzed graphs and use the previously derived and compiled Boolean equations. Even when all channel graphs have been previously analyzed so no new equations must be derived and compiled, the analysis time required to check for this equivalence and generate the final networks of Boolean equations may be substantial.

The advantages and disadvantages of the compiled and interpreted approaches to switch-level simulation suggest a mixed approach. The challenge is to be able to achieve the advantages of both approaches while lessening the disadvantages and to do so in a single simulator. There are two goals to strive for in such a hybrid scheme: fast simulation time and fast setup time. Total simulation time is minimized by using the compiled approach of COSMOS on small channel graphs where the size of the equations is relatively small. The MOSSIM algorithm is used on complex graphs (where the size of the equations can be prohibitive) as well as on certain large graphs which exhibit very localized activity.

The hybrid strategy can also be beneficial in reducing the setup time for large circuits. By a decomposition of large circuits into modules and use of the interpreted approach for the channel graphs that cross module boundaries, the setup required for the simulation can be reduced by a factor of 3. One can think of this approach as analogous (and in fact equivalent) to incremental compilation of object modules and relinking into run-time executables. By building compiled representations of modules individually, and simply relinking them at run-time, significant time savings can be obtained in the modify-simulate loop.

Our hybrid scheme was implemented by modifying the front-end to COSMOS and exploiting its behavioral modelling capability. All interpreted channel graphs are not compiled but instead are included in a data structure placed in a behavioral module that implements the MOSSIM-II algorithm. The advantages of this scheme for reducing simulation time were verified with a series of experiments on common circuit modules such as memories and shifters. Several large designs were used to demonstrate the improvements in setup times.

High-level Timing Specification, Simulation, and Synthesis

Tod Amon and Gaetano Borriello

The largest CAD effort under the contract involved timing issues in the specification, simulation, and synthesis of digital circuits. The specification of a digital design encompasses behavioral and structural aspects with both functional and timing components. A modern design representation must be able to capture elements in the entire space in order to support high-level synthesis, simulation, and verification tools. We proposed a new model called operation/events graphs that meets these criteria. The model relies on the separation of functional and timing elements into a bipartite graph that is a hybrid of data-flow and event-graph models. Special focus has been placed on a general mechanism for timing specification using a restricted first-order predicate calculus. The model is interesting in that it has a clear semantics that make it a solid foundation for a complete set of high-level tools some of which have been implemented or are under development. [AmBo90, AmBo91a, AmBo91b, AmBo91c, AmBo91d, AmBoSe90, AmBoSe91, AmBoSeWi89, BoDe88a, BoDe88b, Bo88, Bo89, Bo90]

Circuit behavior is the high-level description of what a circuit does without overly specifying how that computation is performed. Circuit structure is the low-level description of how the computation is implemented, that is, the logic gates and flip-flops used. Functional aspects of both behavior and structure describe the data transformations and computations to be performed on the inputs in order to generate the outputs. In contrast, timing relationships for behavior and structure describe temporal properties such as minimum and maximum separation times for signal events.

Integration of the entire space of design representation into a unified framework is critical to tool development. Incremental synthesis algorithms can be employed if a design can be evolved from behavior to structure in a single representation. A single simulator can be used to simulate the design at any point in the synthesis process and validate the tools and specification. Verification tools can analyze timing and functional requirements and report their satisfaction. Domain specific languages can be used to enter design descriptions which compile into the underlying representation.

Existing representation methods focus on functional aspects with only rudimentary capabilities for timing information (only structural timing). Specification methods provide rich models for the transformation of data while assuming a fixed and simple timing model (usually, a single-phase synchronous clocking scheme). High-level synthesis tools employ this model to transform functional behavior into a circuit structure that performs the same data transformations in the same partial order. Simulators exist that designers can use to empirically verify that what was specified is what was desired. Verification tools can demonstrate the equivalence of behavioral and structural functionality.

Why is this state of specification, simulation, synthesis, and verification tools not adequate? These tools enable us to automatically synthesize many interesting circuits from high-level specifications. Why should a design representation also include timing behavior? The reasons are many.

First, when synthesizing a circuit not all aspects of its environment are under the control of the designer. That is, the circuit must conform to the environment in which it will be placed. The environment may demand that particular timing relationships be respected. These can be as simple as setup and hold times or as complex as the spacing between messages sent over a local area network. Second, even within a circuit we must be able to adequately describe the timing methodologies used to implement different parts of the circuit. These include precharging constraints, pipeline interlocks, and multiple-phase clocks. Last, we must be able to provide an abstraction of a circuit based on its interface behavior. This is important for information hiding, that is, the ability to specify the essential aspects of a circuit's behavior while minimizing the description of its internal realization. Also, we must have the ability to describe interfaces that may not have a circuit realization at all, for example, the specification of a system backplane bus protocol and the timing constraints that must be respected for proper operation.

The representation we have proposed is a straightforward graph model whose basis consists of two types of nodes connected by a directed arc to form a bipartite graph. Both types may be hierarchical. Event nodes represent changes in logic level on circuit wires and are used to express timing properties of behavior and structure. Operation nodes correspond to the functional aspects of behavior and structure. Boxes contain program code that is evaluated whenever an input event occurs (e.g., C++ source code). The evaluation may conditionally generate output events which will occur at some future point in time and/or possibly change internal state. An example of an operation node is a logic gate that may generate an output event whenever an input event occurs. The delay in generating the output event corresponds to the propagation delay of the gate. A more abstract example of a box is one that forks two independent processes: an input event arrives at an operation node that will then cause two parallel output events thereby permitting two parts of the specification to proceed in parallel. The events, in this case, do not correspond to logic transitions and instead represent abstract control flow. Dependency arcs specify the flow of events in to and out of boxes. The graph is bipartite because dependency arcs specify flow of control by directing events into boxes and the output of boxes to events. We purposely make no distinction between data and control to permit optimization and synthesis algorithms full flexibility in making or not making the distinction.

Although simple timing constraints have an obvious representation (i.e., a labelled directed edge in the graph representing the difference in time between two events) the representation of constraints between events nested within loops, forks, conditional branches, and concurrent structures requires a more comprehensive mechanism. The problem is that constraints are relationships between discrete events (individual occurrences of events) not event nodes.

Identifying discrete events is problematical in the general case. Chronological relationships can often be used to identify the discrete events involved. However, constraints may also be relative to a particular execution path in a complex graph. We have developed the concept of event ancestry to address this issue. Event ancestry is specified by naming the events and internal state that are used to generate an output event. Every discrete event has an ancestry tree, consisting of its immediate ancestors and their ancestors, transitively. Intuitively, the ancestors of an output event are the input events that were used to determine whether to generate the output event.

A first-order predicate calculus is then used to relate discrete events based on the chronological and ancestral relationships described above. Standard Boolean relations and quantifications are augmented with two new primitives to access the time stamp of an event and its ancestry. The full first-order predicate calculus introduces problems which we have addressed by restricting the representation of timing constraints to the following format: (1) universal quantification of the discrete events involved in the constraint, (2) specification of the context within which the constraint must hold and, (3) specification of a particular timing relationship that is required to be true when the context is true.

These simplifications were made for two primary reasons. First, event generation (existence) is represented in the functional components of the graph. Therefore, existential quantification, which is used primarily to constrain functionality, is already encapsulated within our representation. Second, in simulation, the universe of discrete events changes as new events occur and are added to the universe. Constraints are statements about the infinite universe of events. If the universe is constantly changing, when can constraints be checked and violations reported?

We have implemented a simulator for OEgraphs, called OEsim, that not only performs a functional simulation of the circuit but also incrementally checks for timing constraint violations as events occur. Violations can be detected and reported for constraints with universal quantification because a particular instantiation of discrete events causes constraint violation. Many optimizations are performed in the simulator to manage the ever growing ancestry trees of events. Pruning of the trees is performed at run-time and compile-time based on an analysis of the timing constraints and determining when events are no longer needed, that is, they will never be involved in a future constraint violation. The simulator is written in C++ and compiles an OEgraph description into an executable. This leads to an efficient simulator as well as providing the capability to insert arbitrary code in operation boxes. This flexibility can be used to model complex environments, use the file systems to retrieve and store events, and instrument the specification with special-purpose monitors or user-interface code.

We are now turning to the problems of synthesis using OEgraphs. Our focus is on circuit interfaces. In one effort, we are continuing earlier work that dealt with the automatic synthesis of interface glue logic that connects an existing circuit to its environment. This circuitry is

mostly control and may include synchronous as well as asynchronous components. Furthermore, timing constraints on interface signalling protocols must be respected for the circuit to be able to communicate. A second effort is just beginning to look at how interfaces influence the synthesis of the circuit. For example, stringent timing constraint may require a higher-degree of parallelism in the circuit than may have been otherwise economical.

Our synthesis work is leading us to a model of synthesis where circuit functionality is described free of timing considerations. All timing related considerations (signalling, throughput, cycle time, latency, etc.) are specified with the interfaces of the circuit. In this manner, as interfaces change (corresponding to a new environment for the circuit) the synthesis algorithms will also generate a different circuit optimized for the new conditions. It is this modularity of specification and timing-driven synthesis that is the objective of our work in this area.

Parallel Simulation

Mary Bailey

Simulation is unquestionably the most time consuming part of the computationally intensive task of circuit design. Parallel simulation has been frequently suggested as a means of speeding up simulation. Towards that end, Mary Bailey [Ba89a, Ba89b, SaSn89] conducted an extensive analysis of accelerating circuit simulation by means of parallelism. She discovered two principal results. First, using direct experimentation, she showed that there is very little parallelism in event-based logic level simulators. This was greeted by considerable surprise in the community since most researchers assumed circuit simulation to be very parallel. Second, of the various methods available for logic level simulation, the greatest speedup in the synchronous case is obtained by unit-delay strategy.

Each topic will be considered separately.

Empirical Results

The intuition that leads one to conclude (erroneously) that there is much parallelism in circuit simulation runs as follows: Chips have many rapidly switching transistors that "run" in parallel; these should be able to be simulated concurrently. Concurrent execution can be easily achieved by assigning one or a few transistors to each processor. The parallelism of the simulation is thus related to the concurrency of the transistor switchings on the chip.

To test this intuition, we instrumented our standard RNL simulator to keep track of the number of events that execute concurrently for a given chip, i.e. the transistor's switching activity starts at the same time. An event is essentially the evaluation of the state of one or a pair of transistors. The nine chips used for the comparison came from designs done at UW and ranged in size from a decoder with 208 transistors to the Apex spline curve generator (see the Section on Apex) with 61,660 transistors. Since most designs had many thousands of transistors, it was assumed that the parallelism would be measured in the thousands. The results, however, were astounding.

The average parallelism ranged from a high of 25 for the 1,536-transistor 8-stage, 16-bit shift register (which was purposely included in the test to have a large amount of parallelism) to a low of 2.8 for the 2,162-transistor 8x8 Baugh-Wooley multiplier. The Quarter Horse, a 24,068-transistor RISC processor, had an average of 6.3 transistors switching at once. (The Apex measured 23.) Since these numbers assume an unlimited processors and no communication overhead, it would clearly be difficult to speed simulation 100-fold based on so little transistor switching.

These astonishingly low numbers can be explained by looking at the maximums and mini-

mums. The maximum number of concurrent events tended to be quite large, being 69 for the shifter, 22 for the multiplier, 160 for the Quarter Horse, and 520 for the Apex. However, to have such low averages, it was necessary to have a substantial amount of the simulation be "sequential", where only one event was occuring at once. The percents of sequential activity were, respectively, 3.4%, 48%, 40% and 15%. What these data are apparently describing are simulations when many events happen in a burst, e.g. as a result of a clock edge, followed by a large number of subsidiary events that happen at different times and are thus unlikely to coincide with other events. This motivates some of the studies in the next section.

There are explanations for some of this data based on what the circuits do. The RISC processor is a good example. Large parts of the design, most of the register arrays, the shifter, etc., are not intended to switch all of the time. These components are used selectively, being quiescent most of the time. This will lead to low average parallelism. Also, the Apex achieved its 23-fold average parallelism from four copies of a vertex node (5.2 parallelism) and three copies of a processor node (8.0 parallelism). Though the parallelism numbers don't "add" because of dependencies, large grain concurrency does increase the fine grain concurrency. This is especially obvious in the sequential activity, where the Apex had only 15% as compared with 38% and 24%, respectively, for the components.

Bailey conducted a wide range of additional experiments that were described in her dissertation [Bailey 89b].

Understanding Parallel Circuit Simulation

Having observed that there is little parallelism on a chip and explaining why that might be naturally leads one to consider how to improve circuit simulation with parallelism. Bailey conducted such a study and found that two components figure into the performance of a simulator: timing model and synchronization policy.

Timing Model. To say that the transistors on the chip start and finish switching at a particular instant in time is, of course, an over simplification. Each simulator models the continuous charging and discharging of nodes differently. These differences influence not only the accuracy of the simulation but also its parallelism.

Bailey analyzed three different timing strategies: variable delay, fixed delay, and unit delay. Each concerns how much time is to be assigned to the activity of switching the node. In variable delay, the time depends on the topology of the circuit and characteristics of the current state of the node. In principle, there are an infinite number of different delays possible for transistors of the circuit. RNL has this characteristic. In fixed delay models, a small representative set of these times are selected and used exclusively. Typically, the elements of the set differ as to whether the gate is rising or falling. Lsim has this character. Finally, there is the unit delay model where all transitions are assumed to take the same

amount of time. MOSSIM II and COSMOS are instances of unit delay.

The timing model is important because it determines the likelihood that transistors that are physically switching at the same time will be so modeled by the simulator. Clearly, it is unlikely in the variable delay model that two transistors start switching at the same time, while in the unit delay model it is assured, since all of the successors of "simultaneous" events will also be "simultaneous." Since we define concurrency in terms of the number of transistors starting at the same time, a variable delay model will exhibit less concurrency than a unit time model. Experiments prove this out.

Synchronization Strategy. Most circuit simulators are designed to have synchronous parallel execution in that they compute all of the transistor switchings happening in the same time step and then perform a barrier synchronization. Processors do not begin computing transistor switchings for the next time step until all activity for the last time step is complete. This causes many processors to be ideal because the transistors they are modeling are either inactive during a time step or took less computation to evaluate. It would clearly be advantageous to "move ahead" of the global clock.

There are two common "asynchronous" strategies. The conservative *Chandy-Misra* technique permits each processor to advance the clock as far forward as is "safe," i.e. the "future" time of all inputs is known and cannot change. Optimistic strategies, pioneered by *Jefferson*, move forward keeping track of intermediate state so that if they "get too far ahead" they can undo the computation back to an earlier time.

Bailey is able to prove that for the synchronous strategies, the unit delay model gives the greatest amount of parallelism. For a given timing strategy, the asynchronous conservative is the preferred policy when there are an unlimited number of processors. However, for a fixed number of processors, cases can be found where the conservative and the optimistic approaches are each better. Moreover for unit delay, the conservative asynchronous model provides the same speedup as does the synchronous model. Finally, it was observed that the circuit simulation is an instance of the "circuit value problem" studied by theoreticians for years and thought to be "inherently sequential." Bailey's work would tend to be empirical evidence to support this.

WIN descriptions of Circuit Families

Rudolf Nottrott and Wayne Winder

The Washington Integrated Notation (WIN) is a structured environment for developing design generators [BaLiMcNo* 88]. Developed by Rudolf Nottrott and Wayne Winder, WIN is the basis for a set of design generators used extensively within the LIS and distributed as a part of Release 3.2 of the LIS toolset.

A design generator is defined as a process that creates multiple representations of a single instance from a family or class of VLSI circuits. Examples of instances are: 1) a 3-input, 2-phase clock, precharge style decoder, 2) a 16-bit shift register, and 3) a 12-bit by 16-bit precharge style, Booth-encoded, pipelined multiplier. The differences among instances in a class can be contained in concise parameters, such as the bit width of a register or the operand bit width of an arithmetic unit (such as a multiplier). The purpose of WIN is to allow parametric representation of the entire family of circuits in a clear, concise way that emphasizes common elements between representations.

A WIN notation instance, or program, describes one or more representations of a single instance of a circuit family. A different circuit instance from the same family can be described by the same program with one or more changes to the values of parameters.

Central to the purpose of representing circuit families has been the development of WIN interpreters which allow construction of the desired representations from a WIN program. The representations implemented are layout and network. The layout interpreter, *laygen*, creates layout suitable for graphical viewing and subsequent processing (by *magic* [see "1986 VLSI Tools" by Ousterhout, et al. of the University of California, Berkeley, Computer Science Division], for instance). The network interpreter, *netgen*, acts as an input preprocessor for the simulation system *Network C*. The common element to be captured is, essentially, the way in which constituent elements (leaf cells) are put together. WIN allows easy, incremental development of the description of a circuit family, both top-down for circuit design and bottom-up for layout implementation. It also allows the facile capture of design expertise.

Modeling Electronic Circuits and Systems with Network C

William Beckett

Network C, or *nc*, is a superset of the C programming language designed to facilitate the construction of simulation models of electronic circuits and systems. Written by William Beckett, *nc* has been tested on a variety of systems and included in Release 3.2 of the LIS toolset.

The *nc* circuit description constructs of the language are hierarchical and allow subsystem models of varying levels of abstraction to be mixed. The language provides a range of modeling capabilities including complete or approximate solution of Kirchoff equations at the circuit level and discrete event functional simulation at the system level.

Like C, *nc* is a compiled language. System, subsystem, and device models are translated into procedures which are then linked with the *nc* run time library to form an executable simulation model. Since all *nc* models are procedures, libraries of these models can be built which can then be used as components in models of more complex systems.

Following translation, the execution of an *nc* program proceeds in two phases. The first phase is circuit analysis. The effect of circuit analysis is the decomposition of the system being modeled into a set of stages. Each stage is isolated in that there is no DC coupling between it and any other stage. In the case of analog circuits, this decomposition of a circuit into a set of stages is represented explicitly in the language. In the case of MOS integrated circuits, the decomposition is performed automatically and the boundaries of stages need not be explicitly represented.

The second phase of execution of an *nc* program is the calculation phase. The kind of behavior calculation performed by *nc* in this phase depends on the kind of system being modeled. If the system is composed entirely of functional level models, the behavior will be calculated using discrete event simulation. If the system consists entirely of analog circuit level models, a continuous time solution to Kirchoff equations is computed using a non-linear equation solver.

In general, the calculation phase of *nc* will utilize a combination of continuous time calculation and discrete event simulation in which the outputs of each stage are calculated using the equation solver and propagated to the inputs of subsequent stages using discrete event simulation. This technique is aimed at retaining some of the accuracy of purely continuous time systems while realizing the speed advantage inherent in discrete event systems. For models of MOS integrated circuits, the value of increased accuracy over purely discrete systems is that a larger class of circuits can be modeled. This class includes circuits which utilize analog circuit techniques or in which there is a considerable amount of charge sharing. The value of the increased speed of discrete event systems over purely continuous time systems is that

circuits with a larger number of components can be accommodated.

The continuous time part of the calculation used by *nc* is similar in principle to the calculation done in SPICE but differs in several respects. First, since *nc* partitions the circuit into stages and computes each stage independently, the rank of the set of node equations is dramatically reduced for larger systems. Second, the numerical method used by *nc* varies with the kind of circuit being modeled so that the amount of computation may be reduced by taking advantage of special properties of the various types of device models. And finally, *nc* generally uses simple first order device models for transistors and other components. All of these aspects tend to trade accuracy for speed.

Automatic Layout Compaction

Simon Kahan and Martine Schlag

Compaction of VLSI layouts is well studied, computationally intensive and of considerable practical value. An important special case for which our research produced a substantially faster algorithm is the compaction of a design built from a single cell. The entire array can be compacted by compacting a single instance of the cell against itself. Alternatively, the problem may be viewed as finding the smallest area tile enclosing the cell's layout and capable of 4-tiling the plane. The best previously known algorithm was the Mehlhorn and Rülling $O(n^2 logn)$ iterative solution. Anderson, Kahan and Schlag present an $O(nlogn)$ solution for one dimensional, planar constrained networks [AnKaSc89, AnKaSc90]. The two dimensional case is found by iteratively applying the one dimensional solution.

The algorithm was implemented in C in fewer than 500 lines. An effective way to apply the technique is to combine two or more base tiles into a super tile which is compacted internally. This allows components to shift internally making a smaller overall design. The program showed that this technique could be quite powerful [AnKaSc89, AnKaSc90].

Release 3.2 of the LIS Design Tools

One of the spinoffs of the research performed under the contract was the development of software tools to aid IC design. Because of the need for such tools, particularly within Universities, a package of these tools was assembled and distributed. This package contained not only those developed at the University of Washington, but also software developed at various other sites, including UC Berkeley, CMU and MIT. The intent was to provide a reasonably integrated set of tools and documentation that sites could simply load and run, eliminating the time-consuming process of obtaining individual tools from a variety of sites and making them work together.

The entire tools package was ported to and tested on 4 different hosts – SUN3, DEC VAX (running ULTRIX), IBM RT (running Berkeley Unix 4.3) and Sequent. Some of the tools in the toolset and their authors are:

- Network C (William Beckett). A multi-level simulation system designed for constructing and simulating models of VLSI circuits and systems. The input language, a superset of C, supports a range of modeling capabilities including solution of Kirchoff equations at the analog level and discrete event simulation at the system level.
- Gemini (Carl Ebeling). A circuit comparison program that is used to compare extracted circuit layout with a specification.
- Ohmics (Wayne Winder). Checks CMOS designs for the adequacy of ohmic contacts. The output of the circuit extractor Mextra is analyzed to determine the shortest path from each transistor to an ohmic contact of the correct type.
- WIN (Rudolf Nottrott and Wayne Winder). A specialized circuit design language for assembling layouts and netlists.
- CFL (William Beckett). A library of routines that allows parametrized layouts to be assembled within C programs.
- X11 Support (Warren Jessop). The X11 drivers for the popular layout editor Magic.

Release 3.2 of the LIS toolset was distributed to over 90 sites. Besides the value to the site, the LIS received considerable useful feedback.

References

[AmBo91a] T. Amon, G. Borriello, "A Restricted Calculus for the Specification of Timing Behavior," *the International Workshop on Formal Methods in VLSI Design* (poster), Miami, FL, January 1991.

[AmBo90] T. Amon, G. Borriello, "On the Specification of Timing Behavior," *International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (Tau '90 - ACM/IEEE),* Vancouver, BC, August 1990.

[AmBo91b] T. Amon, G. Borriello, "Sizing Synchronization Queues: A Case Study in Higher Level Synthesis," *5th International Workshop on High-Level Synthesis,* Buhlerhohe, Germany, March 1991.

[AmBo91c] T. Amon, G. Borriello, "Sizing Synchronization Queues: A Case Study in Higher Level Synthesis," *28th ACM/IEEE Design Automation Conference,* San Francisco, CA, June 1991.

[AmBo91d] T. Amon, G. Borriello, "OEsim: A Simulator for Timing Behavior," *28th ACM/IEEE Design Automation Conference,* San Francisco, CA June 1991.

[AmBoSe91] T. Amon, G. Borriello, C. Sequin, "Operation/Event Graphics: A Design Representation for Timing Behavior," *10th International Symposium on Computer Hardware Description Languages,* Marseille, France, April 1991.

[AmBoSe90] T. Amon, G. Borriello, C. Sequin, "Operation/Event Graphs and OEsim," Dept. of Computer Science and Engineering, University of Washington, CSE 89-01-17, January 1990.

[AmBoSeWi89] T. Amon, G. Borriello, C. Sequin, W. Winder, "A Unified Behavioral/Structural Representation for Simulation and Synthesis," *4th International High-Level Synthesis Workshop,* Kennebunkport, ME, October 1989.

[AnArBo89] C. Anderson, K. Armstrong, G. Borriello, "Proceedings of CS568: PHM – A Programmable Hardware Monitor," Dept. of Computer Science and Engineering, University of Washington, CSE 89-09-11, September1989.

[AnKaSc89] R. Anderson, S. Kahan, M. Schlag, "An O(*nlogn*) Algorithm for 1D Tile Compaction," in M. Nagl (ed.), *Graph-theoretic Concepts in Computer Science,* LNCS 411, Springer-Verlag, pp. 287-301, 1989.

[AnKaSc90] R. Anderson, S. Kahan, M. Schlag, Single-Layer Cylindrical Compaction, Technical Report 90-10-11, Dept. of Computer Science, University of Washington, 1990, (to appear in *Algorithmica*).

[BaLiMc88] J.L. Baer, M. Liem, L. McMurchie, R. Nottrott, L. Snyder, W. Winder, "A Notation for Describing Multiple Views of VLSI Circuits," *1988 Design Automation Conference (DAC)*, Anaheim, CA, June 1988, pp. 102-107.

[Ba89a] M.L. Bailey, "How to Measure the Parallelism of CMOS VLSI Circuits," *Progress on Computed Aided VLSI Design*, Vol. 5, 1989.

[Ba89b] M.L. Bailey, "The On-Chip Parallelism of VLSI Circuits," Ph.D. dissertation, TR #89-08-05, Dept. of Computer Science and Engineering, University of Washington, July 1989.

[BaSn89c] M.L. Bailey, L. Snyder, "A Model for Comparing Synchronization Strategies for Parallel Logic-Level Simulation," *Proceedings of ICCAD*, November 1989.

[BaSn88] M.L. Bailey, L. Snyder, "An Empirical Study of On-Chip Parallelism," *Design Automation Conference (DAC)*, Anaheim, CA, June 1988, pp. 160-165.

[BaSn89a] M.L. Bailey, L. Snyder, "The Effect of Timing on the Parallelism Available for Parallel Circuit Simulation," in *Distributed Simulation, 1989*, pp. 157-163, SCS, 1989.

[BaSn89c] M. Bailey, L. Snyder, "Towards a Unified Theory of the Parallelism Available for Circuit Simulation," TR #88-11-08, Dept. of Computer Science and Engineering, University of Washington, 1989.

[BaBoRa89] K. Bartlett, G. Borriello, S. Raju, "Timing Optimization of Multi-Phase Sequential Logic," TR #89-12-05, Dept. of Computer Science and Engineering, University of Washington, December 1989.

[BaBoRa90] K. Bartlett, G. Borriello, S. Raju, "Timing Optimization of Multi-Phase Sequential Circuits," *Hawaii International Conference on System Sciences (HICSS23)*, Kailua-Kona, HI, January 1990.

[BaBoRa91] K. Bartlett, G. Borriello, S. Raju, "Timing Optimization of Multi-Phase Sequential Logic," *IEEE Transactions on CAD*, Vol. 10, No. 1, January 1991.

[BeEbWiDe91] R. Bedichek, C. Ebeling, G. Winkenbach, T. DeRose, "Rapid Low-Cost Display of Spline Surfaces," in C. Sequin (ed.), *Proceedings of the Conference on Advanced Research in VLSI*, MIT Press, pp. 340-355, March 1991.

[BoSn91] K. Bolding, L. Snyder, "Performance Study of Two Dimensional Chaotic Routers," in preparation, 1991.

[Bo88] G. Borriello, "Combining Event and Data-Flow Graphs in Behavioral Synthesis," *International Conference on Computer-Aided Design (ICCAD)*, Santa Clara, CA, pp. 322-325, November 1988.

[Bo89] G. Borriello, "Synthesis of Asynchronous/ Synchronous Control Logic," *Proceedings of the 1989 International Symposium of Circuits and Systems (ISCAS'89)*, Portland, OR, May 1989.

[Bo90] G. Borriello, "Synthesis of Interface Logic," *Proceedings of the IEEE VLSI Workshop*, Tampa, FL, February 1990.

[BoDe88a] G. Borriello, E. Detjens, "High-Level Synthesis: Current Status and Future Directions," *1988 Design Automation Conference (DAC)*, Anaheim, CA, June 1988, pp. 477-482.

[BoDe88b] G. Borriell, E. Detjens, "High-Level Synthesis: Current Status and Future Directions," Dept. of Computer Science, University of Washington, CS 88-07-01, July 1988.

[BoEb89] G. Borriello, C. Ebeling, "A One-Year Graduate Course Sequence in VLSI Design and CAD," *Proceedings of the Second VLSI Education Conference*, Santa Clara, CA, July 1989.

[BoKa87] G. Borriello, R. Katz, "Synthesis and Optimization of Interface Transducer Logic," *International Conference on Computer-Aided Design (ICCAD)*, Santa Clara, CA, November 1987.

[BuBo89] S. Bunton, G. Borriello, "Development of a Theme by Ziv and Lempel," Dept. of Computer Science and Engineering, University of Washington, CSE 89-10-08, October 1989.

[BuBo90] S. Bunton, G. Borriello, "Practical Dictionary Management for Hardware Data Compression, *6th MIT Conference on Advanced Research in VLSI*, Boston, MA, April 1990.

[BuBo91] S. Bunton, G. Borriello, "Practical Dictionary Management for Hardware Data Compression," *Communications of the ACM*, accepted in July 1990, expected to appear in July 1991.

[BuSaBaBo91] T. Burks, K. Sakallah, K. Bartlett, G. Borriello, "Performance Improvement through Optimal Clocking and Retiming," *International Workshop on Logic Synthesis* (poster), Research Triangle Park, NC, May 1991.

[CaBo89] J. Carson, G. Borriello, "A Testable CMOS Asynchronous Counter," Dept. of Computer Science and Engineering, University of Washington, CSE 89-10-07, October 1989.

[CaBo90] J. Carson, G. Borriello, "A Fully Testable CMOS Asynchronous Counter," *IEEE Journal of Solid-State Circuits,* Vol. SC-25, No. 4, August 1990.

[DeBaBa*89] T. DeRose, M. Bailey, B. Barnard, R. Cypher, D. Dobrikin, C. Ebeling, S. Konstantinidou, L. McMurchie, H. Mizrahi, B. Yost, "Apex: Two Architectures for Generating Parametric Curves and Surfaces," *The Visual Computer,* Vol. 5, No. 3, June 1989.

[Eb88] C. Ebeling, "GeminiII: A Second Generation Circuit Comparison Program," *Proceedings of the International Conference on Computer-Aided Design 88,* 1988, pp. 322-325.

[Eb89] C. Ebeling, "GeminiII: A Second Generation Layout Validation Program," *Proceedings of ICCAD,* pp. 322-325, November 1989.

[Eb91] C. Ebeling, "Flexible Hardware for Computing Complex Evaluation Functions," Technical Report, Dept. of Computer Science and Engineering, University of Washington, 1991.

[EbBo90] C. Ebeling, G. Borriello, "Making the Most of a Design Project," *Proceedings 1990 Microelectronic Systems Education Conference,* 1990.

[EbMc90] C. Ebeling, N. McKenzie, "MacTester: A Low-Cost Tester for Interactive Testing and Debugging," *Proceedings 1990 Microelectronic Systems Education Conference,* July 1990.

[EbWu89a] C. Ebeling, Z. Wu, "WireLisp: Combining Graphics and Procedures in a Circuit Specification Language," *Proceedings of IEEE International Conference on CAD (ICCAD-89),* November 1989.

[Eb89b] C. Ebeling, Z. Wu, "WireLisp: Graphical Programs for Digital System Design," *Proceedings of the International Conference on Computer Aided Design,* November 1989.

[HoSn90] S. Ho, L. Snyder, "A Formal Model of the Processor Memory Interface," *Proceedings of the International Conference on Parallel Processing,* Vol. 1, Penn State, pp. 181-184, 1990.

[HoSn88] S. Ho, L. Snyder, "A Model for Architectural Comparison," Dept. of Computer Science, University of Washington, TR #88-04-01, 1988.

[HoSn89a] S. Ho, L. Snyder, "Are Bit Serial or Word Parallel Computers Faster," Dept. of Computer Science and Engineering, University of Washington, 1989.

[HoSn89b] S. Ho, L. Snyder, "Balance in Architectural Design," Technical Report #89-12-04, Department of Computer Science and Engineering, University of Washington, December 1989.

[HoHoSn89] S. Ho, T. Holman, L. Snyder, "Normalized Time and its Use in Architectural Design," *Proceedings of 27th Allerton Conference on Control, Communications and Computing*, pp. 712-713, September 1989.

[Ko90] S. Konstantinidou, "Adaptive, Minimal Routing in Hypercubes," *Sixth MIT Conference on Advanced Research in VLSI*, Boston, MA April 1990.

[Ko91] S. Konstantinidou, "Deterministic and Chaotic Adaptive Routing in Multicomputers," Ph.D. dissertation, Dept. of Computer Science and Engineering, University of Washington, 1991.

[KoSn90] S. Konstantinidou, L. Snyder, "The Chaos Router: A Practical Application for Randomization in Network Routing," *Proceedings of 2nd Symposium on Parallel Algorithms and Architectures*, ACM, pp. 21-30, 1990.

[McAnBo91] L. McMurchie, C. Anderson, G. Borriello, "Hybrid Compiled/Interpreted Simulation on MOS Circuits," *2nd European Design Automation Conference*, Amsterdam, February 1991.

[Mc90] N. McKenzie, "UW VLSI Chip Tester," TR #89-12-01, Dept. of Computer Science and Engineering, University of Washington, December 1990.

[Sn88] L. Snyder, "A Taxonomy of Synchronous Parallel Machines," *Proceedings of the International Conference on Parallel Processing*, 1988, pp. 160-165.

[Ty87] A. Tyagi, "Hercules: A Power Analyzer for MOS VLSI Circuits," *Proceedings of the International Conference on Computer-Aided Design, ICCAD*, 1987, pp. 529-533.

[Ty89] A. Tyagi, "The Role of Energy in VLSI Computations," Ph.D. dissertation, Dept. of Computer Science and Engineering, University of Washington, 1989.

[Ty88] A. Tyagi, "The Role of Energy in VLSI Computations," UW Dept. of Computer Science, TR #88-06-05, 1988.

[WuEb89a] Z. Wu, C. Ebeling, "WireLisp Manual," TR #89-12-02, Dept. of Computer Science and Engineering, University of Washington, 1989.

[WuEb89b] Z. Wu, C. Ebeling, "Drawing WireLisp," TR #89-12-03, Dept. of Computer Science and Engineering, University of Washington, 1989.